**CIS 41B - Lab 2: numpy, matplotlib, tkinter**

Write a data analysis and visualization GUI application that works with monthly rainfall in San Francisco.

**Input file**

The input file is sf\_rainfall.csv, which has the years and monthly rainfall in San Francisco, from 1850 – 2024 ([source](https://ggweather.com/sf/monthly.html)).

Here are the first 4 lines of the file, shown in table format so it’s easier to see, instead of the comma-separated format in the file.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1850 | 8.34 | 1.77 | 4.53 | 0.46 | 0 | 0 | 0 | 0 | 0.33 | 0 | 0.92 | 1.05 |
| 1851 | 0.72 | 0.54 | 1.94 | 1.23 | 0.67 | 0.02 | 0 | 0.02 | 1.03 | 0.21 | 2.21 | 7.1 |
| 1852 | 0.58 | 0.14 | 6.68 | 0.26 | 0.32 | 0 | 0 | 0 | 0 | 0.8 | 5.31 | 13.2 |
| 1853 | 3.92 | 1.42 | 4.86 | 5.37 | 0.38 | 0 | 0 | 0.04 | 0.46 | 0.12 | 2.28 | 2.32 |

The first field is the year, and the next 12 fields are the monthly rainfall for each month of that year, Jan-Dec.

**Overview**

Lab 1 consists of 2 files that you will turn in: rainfall.py and lab2.py.

* rainfall.py has a Rainfall class, which contains data from the input file and methods to analyze and plot the data.
* lab2.py has a GUI class that interacts with the user and calls the methods in Rainfall class to process the user choice.

The user has 4 ways to see rainfall data:

1. The highest yearly rainfall and year, lowest yearly rainfall and year, and median yearly rainfall of all the years.
2. A plot of the distribution of all monthly rainfall amounts in all the years.
3. A plot of the average rainfall of each month across the years
4. A plot of the yearly rainfall for a range of years.

**rainfall.py**

Create a Rainfall class that has methods to do the following:

1. Initialize the Rainfall object by reading data from the input file and storing the data in one or more numpy arrays.
   * The input file should be opened and read one time only.
   * The last row of data (for the year 2024) only has data for the first 2 months. Don’t include this row in any calculation or plot since the data is not complete.
   * Initialize any instance attributes as needed. This includes finding the highest yearly rainfall and year, the lowest yearly rainfall and year, and the median yearly rainfall of all the years.
2. Plot the monthly rainfall distribution.
   * + Make a copy of the numpy array of rainfall and turn this copy into a 1D array. Make sure you only use the methods we’ve discussed in class, so you have practice with them.
     + Choose the appropriate plot that shows the distribution of all the monthly rainfalls.
     + The plot should have a descriptive title, and x-axis and y-axis labels that explain what the x and y data are.
     + Return the number of monthly rainfalls being plotted. This number should not be hard coded.
3. Plot the average rainfall for each month.
   * + Find the average rainfall for each of the 12 months.
     + Choose an appropriate plot that makes it easy for the user to visually compare the average rainfall of each month.
     + The plot should have a descriptive title, and x-axis and y-axis labels that explain what the x and y data are.
     + Return the number of monthly averages being plotted. This number should not be hard coded.
4. Plot the yearly rainfall for a range of years.
   * + Accept 2 input arguments: a start year and an end year. You can assume the start and end years will be within the valid range of 1850-2023, and that start year < end year.
     + Find a subset of the input years that’s between the start year and end year, inclusive.
     + Find a subset of the yearly rainfall that’s between the start year and end year, inclusive.
     + Choose a plot that will show the yearly rainfall trend from the start year to the end year.
     + Find the average yearly rainfall for all the years between the start year and end year. Should be 1 number.
     + Plot the average yearly rainfall number across all the years, in the same plot of yearly rainfalls.
     + Return the number of years being plotted. This number should not be hard coded.
     + Because matplotlib looks at the range of data to determine the xtick and ytick values, if the range of years is an odd number, such as 2001-2013, then the xticks will be floating point numbers such as 2012.75 because matplotlib tries to evenly space out the tick marks.  
       To ask matplotlib to display integer values for the years, add the following steps:

1. Import this matplotlib module: from matplotlib.ticker import MaxNLocator

2. When configuring your plot, such as plt.title() and plt.grid(), also add:

plt.gca().xaxis.set\_major\_locator(MaxNLocator(integer=True))

For all 3 plots (parts b, c, d), it’s good practice to try different colors for your plot data, instead of relying on the default blue color.

In addition to the Rainfall class, write a decorator that prints the return value of the function that it decorates.

Apply the decorator to the 3 plot methods, so it will print the number of data points being plotted.

Unit testing  
It’s highly recommended that you write the testing code for the Rainfall class to confirm that it works, before moving to the lab2.py file. The test code should have 6 lines:

* Create the Rainfall object.
* Print the highest yearly rainfall and year, lowest yearly rainfall and year, and median yearly rainfall.
* Plot the distribution of the monthly rainfall.
* Plot the average monthly rainfall.
* Plot the yearly rainfall for 1850 – 2023.
* Plot the yearly rainfall for 2000 – 2013.

Lab 2 EC

To really encourage you to do unit testing (a good practice in real life), there will be *2 pts EC* if you show me your code and demo the 6 lines of test code running successfully, before or after class, on Tues 4/30.

**lab2.py**

This file contains 3 classes: a main window class, a dialog window class, and a plot window class. Each of the 3 window classes is derived from an appropriate tkinter class.

The code of the 3 window classes should observe OOP standard practices:

* A window class cannot directly access private data of another window class. Use set/get methods instead.
* Each window class should do its own task. For example, the plot window class should call the plot methods of the Rainfall object. The main window class should not call these plot methods.
* Each window class is responsible for its own behavior. For example, when a window needs to close, its window class should close the window and not rely on another window class to close it.

The 3 window classes:

1. The main window is an object of the main window class, and it appears when the app first comes up.
   * The window has a title, a line of text to explain the purpose of the application, 3 buttons, and the 3 yearly rainfall statistics.
   * The 3 buttons are for the user to see: the monthly average rainfall, the distribution of monthly rainfall, and the yearly rainfall for a range of years.
   * The buttons are placed side by side.
   * The 3 statistics are displayed on 3 lines, with text explanation. Each number should have 2 digits after the decimal point.
   * The following is a sample main window. Feel free to change the wording of the text strings, and the font size/type/color as you like. Also feel free to change text for the buttons.

A screenshot of a computer

Description automatically generated

* When the user clicks on the ‘Monthly Average’ button or ‘Monthly Range' button: the main window creates a plot window with the appropriate plot.
  + When the user clicks on the ‘Yearly Total’ button, the main window creates a dialog window to ask the user for a range of years (2 input numbers). When the user has entered the 2 input numbers, the dialog window closes and the main window creates the plot window with the appropriate plot.
  + When the user clicks X to close the main window, all other windows of the app should also close.

1. The plot window is an object of the plot window class. The plot window is created by the main window:
   * The plot window must be a tkinter window that works with matplotlib, and not an independent matplotlib window. You can tell it’s a plot in a tkinter window if the top of the window shows the tk symbol:

A screenshot of a computer

Description automatically generated

* + There should be one plot window class that can display all 3 plots of the Rainfall class. Do not create different plot window classes.
  + Do your best to not duplicate code in this class, since the code for the 3 Rainfall plots are very similar.
  + The user can click X to close the plot window, or the user can leave the window open and go to the main window to select another choice. This means there can be multiple plot windows opened if the user chooses to keep them open.

1. The dialog window is an object of the dialog window class. The window is created by the main window when the user selects the ‘Yearly Total’ button.
   * The window has a prompt to ask the user to enter a range of years as 2 numbers, and it has a tk.Entry object for the user to enter the 2 numbers.

A screenshot of a computer

Description automatically generated

* + When the dialog window opens, all other windows should be disabled so that the user cannot use the main window to select another choice.
  + After the user has entered the 2 numbers, check for valid user input:
  + There should be two 4-digit numbers, separated by 1 or more space.
  + Starting and trailing spaces are okay.
  + The two 4-digit numbers should be within the range of the first and last years. You should not hard code the years. When I test your code, the input file might have different first and last years.
  + The ending number should be less than the starting year.
  + If the 2 numbers are valid, they are saved by the dialog window object.
  + If the 2 numbers are not valid, a messagebox window pops up to let the user know that the range is not valid, and that the full range (from first year to last year) will be used. Again, do not hard code the first and last years.

A screenshot of a computer error

Description automatically generated

The user will need to click OK to close the messagebox window before the app continues.

* + For both cases (valid or not valid range), execution goes back to the main window, and the main window creates a plot window to plot the yearly rainfall.
  + If, at the dialog window, the user clicks X to close the dialog window instead of entering a range of years, then the user is back at the main window, and no plot is shown.

**Exception handling**

* During GUI start up, data will be read in from the file. If the file open is not successful, a messagebox window lets the user know that there is a file open error, with the specific file name.

A screenshot of a computer error

Description automatically generated

* When the user closes the messagebox window, the main window closes and the application terminates.

**Documentation**

- At the top of each file: put your name, lab number, and module name.

- For each public method: add a docstring

You don't need to add docstrings for private method or 'get' method where the method only returns a data attribute, but you're welcome to do so.